



Programación Orientada a Objetos

Víctor Hugo Saldívar Carrillo

Proyecto Integrador: Calculadora de Matrices

Integrantes:

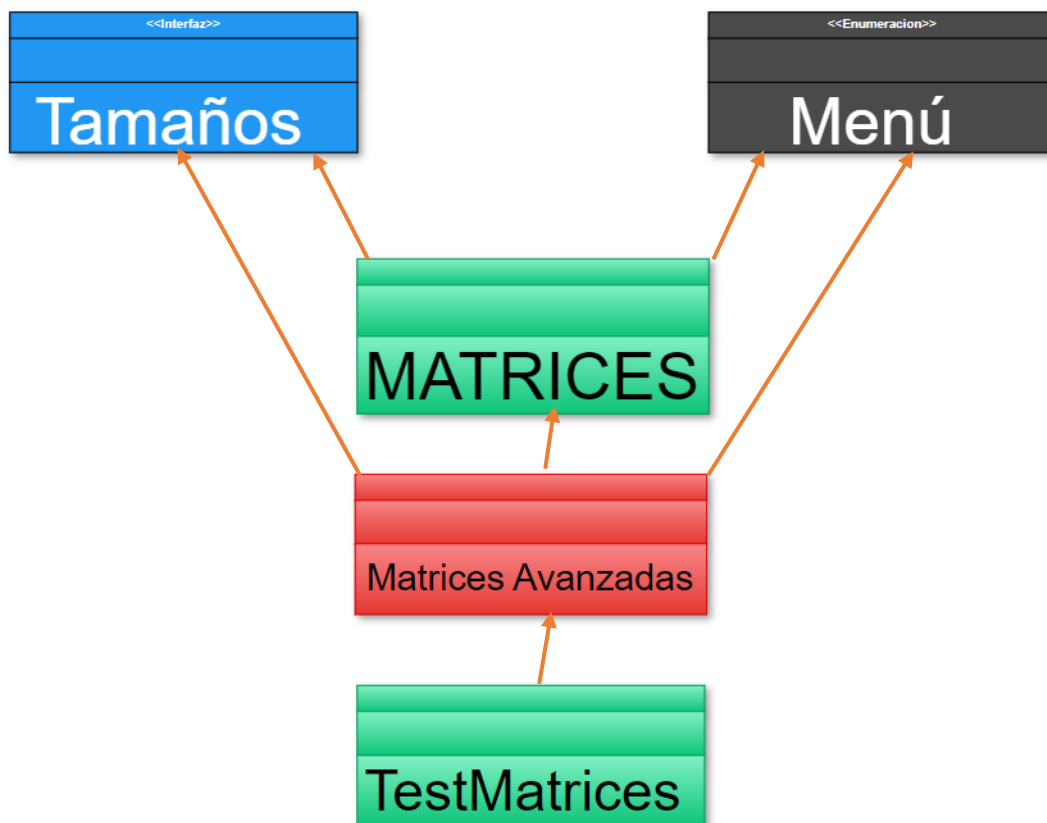
José Alfredo Calvillo Gómez - 733630

Dion Rizo Velarde - 734757

Introducción

Para este proyecto de una calculadora de matrices nosotros decidimos poner a prueba lo que hemos aprendido a lo largo del semestre, las habilidades que pretendemos poner a prueba son las habilidades de la programación orientada a objetos como la herencia de clases, las enumeraciones, los métodos estándares, los métodos y atributos con modificaciones, etc. Esta API será sumamente útil para cualquier persona que desee hacer operaciones con matrices para no tener que hacerlo manualmente ya tiene este programa que lo ayudara a hacerlo por él. Esta API ayudará mucho al usuario ya que tiene una interfaz amigable con el usuario y puede crear las matrices que él quiera, cuando quiera, las puede cambiar en el momento que el deseé, así que esta API se podría decir que es practica y fácil de usar.

Diagramas UML Vista 0



Diagramas UML Vista 1



Pseudocódigo

Algoritmo clase Matrices

Variables doble matriz

entero a, b, filas, columnas, suma

texto mensaje

booleana same

Inicio

Declarar función Matrices

Leer matriz

Para i = 0 and i < filasmatrices aumentar i

Para j = 0 and j < columnasmatrices aumentar j

Leer matriz

Fin función Matrices

Declarar función Suma

Declarar argumentos Matrices m1, Matrices m2

Leer resultante

Declarar variable filas

Declarar variable columnas

Para a = 0 and a < filas aumentar a

Para b = 0 and b < columnas aumentar b

Declarar resultante.matriz

Sumar m1.matriz, m2.matriz

Regresar resultante

Fin función Suma

Declarar función Resta

Declarar argumentos Matrices m1, Matrices m2

Leer resultante

Declarar variable filas

Declarar variable columnas

Para $a = 0$ and $a < \text{filas}$ aumentar a

Para $b = 0$ and $b < \text{columnas}$ aumentar b

Declarar `resultante.matriz`

Restar `m1.matriz`, `m2.matriz`

Regresar `resultante`

Fin función Resta

Declarar función Multiplicación

Declarar argumentos Matrices `m1`, Matrices `m2`

Leer `resultante`

Para $a = 0$ and $a < \text{longitud de } m1.matriz$ aumentar a

Para $b = 0$ and $b < \text{longitud de } m1.matriz$ aumentar a

$\text{suma} = 0$

Para $c = 0$ and $c < \text{longitud de } m1.matriz$ aumentar c

Declarar `suma`

`multiplicar m1.matriz * m2.matriz`

`Resultanto.matriz = suma`

Regresar `resultante`

Fin función Multiplicación

Declarar función Clone

Declarar `obj = null`

Capturar `obj`

Declarar `obj = (Matrices) super.clone`

Gestionar errores

Mostrar en pantalla "¡¡ERROR!!. No se logró clonar la Matriz"

Declarar `obj.Matriz = (double)obj.matriz.clone`

para $i = 0$ and $i < \text{longitud de } obj.matriz$ aumentar i

Implementar i en obj.matriz

Regresar obj

Fin función Clone

Inicio función equals

Declarar argumento Matrices m2

same = true

Si longitud matriz == longitud m2.matriz

Para a = 0 and a < longitud matriz aumentar a

Para b = 0 and a < longitud matriz and same aumentar b

Si matriz != m2.matriz

same = false

Sino same = false

Regresar same

Fin función equals

Declarar función pedirMatriz

Declarar variable filas

Declarar variable columnas

Para a = 0 and a < filas aumentar a

Para b = 0 and b < filas aumentar b

Declarar matriz

Mostrar en pantalla Matriz, Posición

Fin función pedirMatriz

Inicio función toString

Leer mensaje

Para a = 0 and a < longitud matriz aumentar a

Para b = 0 and b < longitud matriz aumentar b

Declarar mensaje = matriz

mensaje = "\n"

Mostrar en pantalla "Resultado" + mensaje

Regresar mensaje

Fin función toString

Inicio función imprimirMatriz

Leer mensaje

Para a = 0 and a < longitud matriz aumentar a

Para b = 0 and b < longitud matriz aumentar b

Declarar mensaje = matriz

mensaje = "\n"

Mostrar en pantalla "Resultado" + mensaje

Regresar mensaje

Fin función imprimirMatriz

FIN clase Matrices

Algoritmo clase MatricesAvanzadas

Variables entero a, b, c, filas, columnas

 doble aux, determinante, escalar

Inicio

Declarar función Transpuesta

Implementar argumento m1

Leer resultante

Declarar filas = filasMatrices

Declarar columnas = columnasMatrices

Para a = 0 and a < filas aumentar a

Para b = 0 and b < filas aumentar b

Declarar aux = m1.matriz

Declarar resultante.matriz = m1.matriz

Declarar `resultante.matriz = aux`

Regresar `resultante`

Fin función Transpuesta

Declarar función Escalar

Implementar argumento `m1, escalar`

Leer `resultante`

Declarar `filas = filasMatrices`

Declarar `columnas = columnasMatrices`

Para `a = 0` and `a < filas` aumentar `a`

Para `b = 0` and `b < columnas` aumentar `b`

Declarar `resultante.matriz`

Multiplicar `m1.matriz * escalar`

Regresar `resultante`

Fin función Escalar

Inicio función Determinante

Implementar argumento `m1`

Leer `resultante`

Imprimir `tamañoDeMatriz`

Declarar `resultante = m1.clone`

Para `c = 0` and `c < tamañoDeMatriz - 1` aumentar `c`

Para `a = 0` and `c + 1` and `b < tamañoDeMatriz - 1` aumentar `b`

Declarar `resultante.matriz`

Multiplicar `resultante.matriz` con argumento `a,c` * `resultante.matriz` con argumento `c, b`

Dividir sobre `resultante.matriz` con argumento `c, c`

Declarar `determinante = 1`

Para `a = 0` and `a < tamañoDeMatriz` aumentar `a`

Multiplicar `determinante * resultante.matriz`

Leer mensaje "Resultado del determinante de Matriz"

Mostrar mensaje en pantalla

Fin de función Determinante

FIN Clase MatricesAvanzadas

Algoritmo clase TestMatrices

Variables entero b, seleccion, error, opc, escalar, seleccionMatrix, matrizSobrecribir
texto mensaje

Inicio

Inicio del Main

Capturar b = 1

matrizA, matrizB, matrizR

Mostrar en pantalla mensaje "Presiona enter para llenar una matriz 3*3"

Leer matrizA.pedirMatriz

Declarar seleccion como messageDialog

Si seleccion == 0

matrizB = matrizA.Clone

Sino matrizB.pedirMatriz

Leer elegirOpcion

Gestionar errores con NumberFormatException con argumento e

Declarar error como messageDialog

Si error == 0

Mostrar mensaje "Saliendo del programa..."

Sino b = 0

Gestionar errores con NullPointerException con argumento f

Declarar error como showDialog

Si error == 0

Mostrar en pantalla como messageDialog

Sino b = 0

Mientras b == 0

Fin del Main

Inicio de la función Menu

Leer opcMenu

Declarar seleccion como ShowDialog

regresar seleccion

Fin función Menu

Inicio función elegirOpcion

Implementamos los argumentos matrizA, matrizB, matrizR

Declaramos opc = 0

Leemos menu desde opc

Declaramos opc como switch mientras opc != 9

Declaramos caso 0

Llamamos Suma

Declaramos caso 1

Llamamos resta

Declaramos caso2

Llamamos multiplicacion

Declaramos caso 3

Declaramos seleccion como ShowDialog

Si seleccion == 0

Declaramos matrizR como transpuesta a matrizA

Sino declaramos matrizR como transpuesta a matrizB

Leemos matriR.toSrtring

Declaramos caso 4

Declaramos seleccion como ShowDialog

Declaramos escalar como parseDouble "Numero para método escalar: "

Si seleccion == 0

Declaramos matrizR como escalar a matrizA

Sino declaramos matrizR como escalar a matrizB

Leer matriz.toString

Declaramos caso 5

Si matrizA.equals a matrizB

Muestra en pantalla "¡Las Matrices son identicas!"

Sino muestra en pantalla "¡Las Matrices NO son iguales!"

Declaramos caso 6

Declaramos seleccion como ShowDialog

Si seleccion == 0

Declaramos MatricesAvanzadas como determinante de matrizA

Sino declaramos MatricesAvanzadas como determinante de matrizB

Declaramos caso 7

Declaramos seleccionMatrix como ShowDialog

Si seleccionMatrix == 0

Declaramos matrizA como imprimirMatriz

Sino declaramos matrizB como imprimirMatriz

Declaramos caso 8

Declaramos matrizSobreescribir como ShowDialog

Si matrizSobreescribir == 0

Declaramos matrizA como pedirMatriz

Sino declaramos matrizB como pedirMatriz

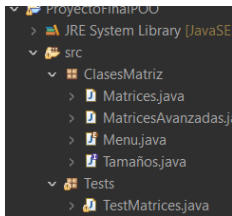
Declaramos caso 9

Muestra en pantalla el mensaje "Saliendo del programa..."

Fin de la clase TestMatriz

Ejemplos de uso de POO en nuestra librería

Clases dentro de paquetes:



Clases con atributos con el modificador adecuado:

```
protected double [][] matriz;
```

Clases con métodos estándares:

```
public boolean equals(Matrices m2) {
    boolean same = true;
    if (matriz.length == m2.matriz.length) {
        for (int a = 0; a < matriz.length; a++) {
            for (int b = 0; b < matriz[0].length && same; b++) {
                if ((matriz[a][b]) != m2.matriz[a][b]) {
                    same = false;
                }
            }
        }
    } else {
        same = false;
    }
    return same;
} //Fin función equals

public String toString() {
    String mensaje = "";
    for (int a = 0; a < matriz.length; a++) {
        for (int b = 0; b < matriz[0].length; b++) {
            mensaje += " " + matriz[a][b] + " ";
        }
        mensaje += "\n";
    }
    JOptionPane.showMessageDialog(null, "Resultado: \n" + mensaje);
    return mensaje;
} //Fin función toString
```

Métodos y atributos con modificadores:

```
public Matrices() {
    matriz = new double [tamañoDeMatriz][tamañoDeMatriz];
    for (int i = 0; i < filasMatrices; i++)
    {
        for (int j = 0; j < columnasMatrices; j++)
        {
            matriz [i][j] = 0.0;
        }
    }
}

//Inicio función Suma
public static Matrices suma(Matrices m1, Matrices m2) {
    Matrices resultante = new Matrices();
    int a, b, filas, columnas;

    filas = filasMatrices;
    columnas = columnasMatrices;

    for (a = 0; a < filas; a++) {
        for (b = 0; b < columnas; b++) {
            resultante.matriz[a][b] = m1.matriz[a][b] + m2.matriz[a][b];
        }
    }

    return resultante;
}
```

Herencias de clases:

```
public class MatricesAvanzadas extends Matrices implements Tamaños{

    //Mandamos a llamar el constructor de la SuperClase Matrices
    public MatricesAvanzadas() {
        super();
    }
}
```

```
@Override
public String toString() {
    return "MatricesAvanzadas [toString()=" + super.toString() + "];"
}
```

Asociación de clases:

```
public class MatricesAvanzadas extends Matrices implements Tamaños{
```

```
public class Matrices implements Tamaños, Cloneable{
```

Interfaces y Enumeraciones:

```
package ClasesMatriz;

public interface Tamaños {
    |


    //Creamos los atributos finales del tamaño de las matrices
    public static final int tamañoDeMatriz = 20;
    public static final int filasMatrices= 20;
    public static final int columnasMatrices = 20;
}
}
```

```
//Creamos la enumeracion
public enum Menu {
    SUMA,
    RESTA,
    MULTIPLICACION,
    TRANSPUESTA,
    ESCALAR,
    COMPARAR,
    DETERMINANTE,PEDIR_MATRIZ,
    REESCRIBIR_MATRIZ,
    SALIR;
}
}
```

Pantalla de ejecución de una aplicación que usa nuestra librería

Selector de opciones

×

 Selecciona una opcion:

SUMA	RESTA	MULTIPLICACION	TRANSPUESTA	ESCALAR	COMPARAR	DETERMINANTE	PEDIR_MATRIZ	REESCRIBIR_MATRIZ	SALIR
------	-------	----------------	-------------	---------	----------	--------------	--------------	-------------------	-------

Como podemos ver en esta captura de pantalla vemos lo que sería siendo el menú principal de la aplicación donde se encuentran todas las opciones que el usuario puede elegir para hacer interacciones con sus matrices.